Available online at www.sciencedirect.com
SCIENCE DIRECT®
Theoretical Computer Science 330 (2005) 205–220
Theoretical Computer Science
www.elsevier.com/locate/tcs

On the size complexity of hybrid networks of evolutionary processors

Juan Castellanos[a], Peter Leupold[b], Victor Mitrana[b, c, *]

[a] *Department of Artificial Intelligence, Polytechnical University of Madrid, Campus de Montegancedo, 28660 Boadilla del Monte, Madrid, Spain*
[b] *Research Group in Mathematical Linguistics, Rovira i Virgili University, Pça. Imperial Tarraco 1, 43005 Tarragona, Spain*
[c] *Faculty of Mathematics and Computer Science, University of Bucharest, Str. Academiei 14, 70109 Bucharest, Romania*

Received 17 November 2003; received in revised form 6 April 2004; accepted 5 July 2004

Abstract

The goal of this paper is twofold. Firstly, to survey in a systematic and uniform way the main results regarding the size descriptional complexity measures of hybrid networks of evolutionary processors as generating devices. Secondly, we improve some results about a size measure, prove that it is connected, and discuss the possibility of computing this measure for regular and context-free languages. We also briefly present a few NP-complete problems and recall how they can be solved in linear time by accepting networks of evolutionary processors with linearly bounded resources (nodes, rules, symbols). Finally, the size complexity of accepting hybrid networks of evolutionary processors recognizing all NP languages in polynomial time is briefly discussed.
© 2004 Elsevier B.V. All rights reserved.

Keywords: Evolutionary operations; Evolutionary processor; Hybrid network of evolutionary processors; Size complexity; Connectivity

* Corresponding author. Faculty of Mathematics and Computer Science, University of Bucharest, Str. Academiei 14, 70109 Bucharest, Romania.
E-mail addresses: jcastellanos@fi.upm.es (J. Castellanos), klauspeter.leupold@estudiants.urv.es (P. Leupold), vmi@fll.urv.es (V. Mitrana).

0304-3975/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2004.07.035

1. Introduction

A rather known architecture for parallel and distributed symbolic processing, related to the Connection Machine [14] as well as the Logic Flow paradigm [9], consists of several processors, each of them being placed in a node of a virtual complete graph, which are able to handle data associated with the respective node. Each node processor acts on the local data in accordance with some predefined rules, and then local data become a mobile agent which can navigate in the network following a given protocol. Only that data which can pass a filtering process can be communicated to the other processors. This filtering process may require to satisfy some conditions imposed by the sending processor, by the receiving processor or by both of them. All the nodes send simultaneously their data and the receiving nodes handle also simultaneously all the arriving messages, according to some strategies, see, e.g. [10,14].

Starting from the premise that data can be given in the form of words, Csuhaj-Varju and Salomaa [7] introduces a concept called network of parallel language processors in the aim of investigating this concept in terms of formal grammars and languages. Networks of language processors are closely related to grammar systems, more specifically to parallel communicating grammar systems [4]. The main idea is that one can place a language generating device (grammar, Lindenmayer system, etc.) in any node of an underlying graph which rewrites the words existing in the node, then the words are communicated to the other nodes. Words can be successfully communicated if they pass some output and input filter. More recently, Csuhaj-Varju and Salomaa [8] introduces networks whose nodes are (standard) Watson–Crick D0L systems which communicate with each other either the correct words or the corrected words.

In [2], we modify this concept in the following way inspired from cell biology. Each processor placed in a node is a very simple processor, a mathematical concept represented by a processor which is able to perform very simple operations, namely point mutations in a DNA sequence (insertion, deletion or substitution of a pair of nucleotides). Such a processor will be called an evolutionary processor. More generally, each node may be viewed as a cell having genetic information encoded in DNA sequences which may evolve by local evolutionary events, that is point mutations. Each node is specialized just for one of these evolutionary operations. Furthermore, the data in each node are organized in the form of multisets of words (each word appears in an arbitrarily large number of copies), and all copies are processed in parallel such that all the possible events that can take place do actually take place. Obviously, the computational process described here is not exactly an evolutionary process in the Darwinian sense. But the rewriting operations we have considered might be interpreted as mutations and the filtering process might be viewed as a selection process. Recombination is missing but it was asserted that evolutionary and functional relationships between genes can be captured by taking only local mutations into consideration [19]. Furthermore, we are not concerned here with a possible biological implementation, though it is a matter of great importance. Consequently, hybrid networks of evolutionary processors might be viewed as bio-inspired computing models.

Our mechanisms introduced in [2] in the aim of solving an NP-complete problem in linear time, are further considered in [3] as language generating devices and their computational power is investigated. Furthermore, filters, based on the membership condition, used in [7]

are generalized in some versions defined in [2,3]. More precisely, the new filters are based on different types of random-context conditions. In the aforementioned papers, the filters of all nodes are defined by the same random-context condition type. Moreover, the rules are applied in the same manner in all the nodes. These restrictions are discarded in [17] and [15]. By this reason, these networks are called *hybrid*.

A similar concept is introduced in [6] inspired by the evolution of cell populations, the one which might model some properties of evolving cell communities at the syntactical level. Cells are represented by words which describe their DNA sequences. Informally, at any moment of time, the evolutionary system is described by a collection of words, where each word represents one cell. Cells belong to species and their community evolves according to mutations and division which are defined by operations on words. Only those cells are accepted as surviving (correct) ones which are represented by a word in a given set of words, called the genotype space of the species. This feature parallels with the natural process of evolution. It is worth mentioning that any recursively enumerable language is a language of a species of an evolutionary system with point mutations of restricted forms. In the aforementioned paper, a connection between Lindenmayer systems (language theoretical models of developmental systems) and evolutionary systems is established, namely the growth function of any deterministic $0L$ system can be obtained from the population growth relation of some (deterministic) evolutionary system.

The paper is organized as follows: in Section 2 we give the definition of an evolutionary processor and of both generating and accepting hybrid network of evolutionary processors (shortly, GHNEP and AHNEP, respectively). Then we discuss the size complexity (number of nodes) of GHNEPs and languages generated by them. We give an upper bound for the size complexity of any language L generated by a GHNEP which is a linear mapping depending on the number of letters appearing in the words of L only. We show that the size complexity is a connected measure and prove that this measure cannot be algorithmically computed for context-free languages. The problem remains open for regular languages; however we propose an algorithm for deciding whether or not the size of a regular language equals 1. In Section 4, we briefly recall some results regarding the size complexity of AHNEPs solving in linear time a few NP-complete problems. The last section briefly discusses the size complexity of AHNEPs accepting all NP languages in polynomial time.

2. Basic definitions

We start by summarizing the notions used throughout the paper. An *alphabet* is a finite and nonempty set of symbols. The cardinality of a finite set A is written $\mathrm{card}(A)$. Any sequence of symbols from an alphabet V is called *word* over V. The set of all words over V is denoted by V^* and the empty word is denoted by ε. The length of a word x is denoted by $|x|$ while the number of occurrences of a letter a in a word x is denoted by $|x|_a$. Furthermore, for each nonempty word x we denote by $alph(x)$ the minimal alphabet W such that $x \in W^*$. We denote by w^R the mirror image of the word w and by L^R the language of mirror images of all words in L. A morphism from $(V \cup U)^*$ to V^* which erases all symbols from U and leaves all symbols unchanged from V is called *projection* and it is denoted by pr_V.

We say that a rule $a \to b$, with $a, b \in V \cup \{\varepsilon\}$ is a *substitution rule* if both a and b are not ε; it is a *deletion rule* if $a \neq \varepsilon$ and $b = \varepsilon$; it is an *insertion rule* if $a = \varepsilon$ and $b \neq \varepsilon$. The set of all substitution, deletion, and insertion rules over an alphabet V is denoted by Sub_V, Del_V, and Ins_V, respectively.

Given a rule σ as above and a word $w \in V^*$, we define the following *actions* of σ on w:

- If $\sigma \equiv a \to b \in Sub_V$, then $\sigma^*(w) = \begin{cases} \{ubv : \exists u, v \in V^* \ (w = uav)\}, \\ \{w\} \quad \text{otherwise.} \end{cases}$
- If $\sigma \equiv a \to \varepsilon \in Del_V$, then $\sigma^*(w) = \begin{cases} \{uv : \exists u, v \in V^* \ (w = uav)\}, \\ \{w\} \quad \text{otherwise.} \end{cases}$

$$\sigma^r(w) = \begin{cases} \{u : w = ua\}, \\ \{w\} \quad \text{otherwise,} \end{cases} \quad \sigma^l(w) = \begin{cases} \{v : w = av\}, \\ \{w\} \quad \text{otherwise.} \end{cases}$$

- If $\sigma \equiv \varepsilon \to a \in Ins_V$, then

$$\sigma^*(w) = \{uav : \exists u, v \in V^* \ (w = uv)\}, \quad \sigma^r(w) = \{wa\}, \ \sigma^l(w) = \{aw\}.$$

$\alpha \in \{*, l, r\}$ expresses the way of applying a deletion or insertion rule to a word, namely at any position ($\alpha = *$), in the left ($\alpha = l$), or in the right ($\alpha = r$) end of the word, respectively. For every rule σ, action $\alpha \in \{*, l, r\}$, and $L \subseteq V^*$, we define the *α-action of σ on L* by $\sigma^\alpha(L) = \bigcup_{w \in L} \sigma^\alpha(w)$. Given a finite set of rules M, we define the *α-action of M* on the word w and the language L by

$$M^\alpha(w) = \bigcup_{\sigma \in M} \sigma^\alpha(w) \quad \text{and} \quad M^\alpha(L) = \bigcup_{w \in L} M^\alpha(w),$$

respectively.

In what follows, we shall refer to the rewriting operations defined above as *evolutionary operations* since they may be viewed as linguistic formulations of local gene mutations. For two disjoint subsets P and F of an alphabet V and a word over V, we define the predicates

$$\begin{aligned} &\varphi^{(1)}(w; P, F) \equiv P \subseteq alph(w) \quad \wedge F \cap alph(w) = \emptyset, \\ &\varphi^{(2)}(w; P, F) \equiv alph(w) \subseteq P, \\ &\varphi^{(3)}(w; P, F) \equiv P \subseteq alph(w) \quad \wedge F \not\subseteq alph(w), \\ &\varphi^{(4)}(w; P, F) \equiv alph(w) \cap P \neq \emptyset \ \wedge F \cap alph(w) = \emptyset. \end{aligned}$$

The construction of these predicates is based on *random-context conditions* defined by the two sets P (*permitting contexts*) and F (*forbidding contexts*). For every language $L \subseteq V^*$ and $\beta \in \{(1), (2), (3), (4)\}$, we define

$$\varphi^\beta(L, P, F) = \{w \in L \mid \varphi^\beta(w; P, F)\}.$$

An *evolutionary processor over V* is a tuple (M, PI, FI, PO, FO), where

- Either $(M \subseteq Sub_V)$ or $(M \subseteq Del_V)$ or $(M \subseteq Ins_V)$. The set M represents the set of evolutionary rules of the processor. As one can see, a processor is "specialized" in one evolutionary operation only.
- $PI, FI \subseteq V$ are the *input* permitting/forbidding contexts of the processor, while $PO, FO \subseteq V$ are the *output* permitting/forbidding contexts of the processor.

We denote the set of evolutionary processors over V by EP_V.

A *generating hybrid network of evolutionary processors* (a GHNEP, for short) is a 7-tuple $\Gamma = (V, G, \mathcal{N}, C_0, \alpha, \beta, x_O)$, where the following conditions hold:

- V is an alphabet.
- $G = (X_G, E_G)$ is an undirected graph with the set of vertices X_G and the set of edges E_G, each edge is given in the form of a set of two nodes. G is called the *underlying graph* of the network.
- $\mathcal{N} : X_G \longrightarrow EP_V$ is a mapping which associates with each node $x \in X_G$ the evolutionary processor $\mathcal{N}(x) = (M_x, PI_x, FI_x, PO_x, FO_x)$.
- $C_0 : X_G \longrightarrow V^*$ is a mapping which identifies the initial configuration of the network. It associates a finite set of words with each node of the graph G.
- $\alpha : X_G \longrightarrow \{*, l, r\}$; $\alpha(x)$ gives the action mode of the rules of node x on the words occurring in that node.
- $\beta : X_G \longrightarrow \{(1), (2), (3), (4)\}$ defines the type of the *input/output filters* of a node. More precisely, for every node, $x \in X_G$, we define the following filters: the input filter is given as $\rho_x(\cdot) = \varphi^{\beta(x)}(\cdot; PI_x, FI_x)$, and the output filter is defined as $\tau_x(\cdot) = \varphi^{\beta(x)}(\cdot; PO_x, FO_x)$. That is, $\rho_x(w)$ (resp. τ_x) indicates whether or not the word w can pass the input (resp. output) filter of x. More generally, $\rho_x(L)$ (resp. $\tau_x(L)$) is the set of words of L that can pass the input (resp. output) filter of x.
- $x_O \in X_G$ is the *output node* of the GHNEP.

An *accepting hybrid network of evolutionary processors* (AHNEP for short) is a 7-tuple $\Gamma = (V, U, G, N, \alpha, \beta, x_I, x_O)$, where:

- V and U are the input and network alphabet, respectively, $V \subseteq U$.
- G, N, α, β, x_O are defined as above, and x_I is the *input node* of the AHNEP.

In the above definitions, we say that $\text{card}(X_G)$ is the size of Γ, denoted by $size(\Gamma)$. If $\alpha(x) = \alpha(y)$ and $\beta(x) = \beta(y)$ for any pair of nodes $x, y \in X_G$, then the network is said to be *homogeneous*. If the set of rules at every node consists of at most one rule, then the network is said to be *elementary*. Further, a network having all filters empty sets is said to be *free*. In the theory of networks some types of underlying graphs are common, e.g., rings, stars, grids, etc. In some of the aforementioned papers [3,5,17,15], there were investigated networks of evolutionary processors having underlying graphs of these special forms, but with a special attention to complete graphs. Thus a GHNEP (AHNEP) is said to be a *star, ring, grid, or complete* GHNEP (AHNEP) if its underlying graph is a star, ring, grid, or complete graph, respectively. The star, ring, and complete graph with n nodes is denoted by S_n, R_n, and K_n, respectively. Most of the results presented in the sequel concern complete GHNEPs (AHNEPs).

A *configuration* of a GHNEP (AHNEP) Γ as above is a mapping $C : X_G \longrightarrow 2^{V^*}$ which associates a set of words with every node of the graph. A configuration may be understood as the sets of words which are present in any node at a given moment. A configuration can change either by an *evolutionary step* or by a *communication step*. When changing by an evolutionary step, each component $C(x)$ of the configuration C is changed in accordance with the set of evolutionary rules M_x associated with the node x and the way of applying these rules $\alpha(x)$. Formally, we say that the configuration C' is obtained in *one evolutionary step* from the configuration C, written as $C \Longrightarrow C'$, iff

$$C'(x) = M_x^{\alpha(x)}(C(x)) \quad \text{for all} \quad x \in X_G.$$

When changing by a communication step, each node processor $x \in X_G$ sends one copy of each word it has, which is able to pass the output filter of x, to all the node processors connected to x and receives all the words sent by any node processor connected with x providing that they can pass its input filter. Formally, we say that the configuration C' is obtained in *one communication step* from configuration C, written as $C \vdash C'$, iff

$$C'(x) = (C(x) - \tau_x(C(x))) \cup \bigcup_{\{x,y\} \in E_G} (\tau_y(C(y)) \cap \rho_x(C(y))) \quad \text{for all} \quad x \in X_G.$$

Let Γ be a GHNEP, a *computation* in Γ is a sequence of configurations $C_0, C_1, C_2, \ldots$, where C_0 is the initial configuration of Γ, $C_{2i} \Longrightarrow C_{2i+1}$ and $C_{2i+1} \vdash C_{2i+2}$, for all $i \geqslant 0$. By the previous definitions, each configuration C_i is uniquely determined by the configuration C_{i-1}. If the sequence is finite, we have a finite computation. The result of any finite or infinite computation is a language which is collected in the output node of the network. For any computation $C_0, C_1, \ldots$, all words existing in the output node at some step belong to the language generated by the network. Formally, the *language* generated by Γ is $L_{\text{gen}}(\Gamma) = \bigcup_{s \geqslant 0} C_s(x_O)$.

Let Γ be an AHNEP, the *computation of Γ on the input word $w \in V^*$* is a sequence of configurations $C_0^{(w)}, C_1^{(w)}, C_2^{(w)}, \ldots$, where $C_0^{(w)}$ is the initial configuration of Γ defined by $C_0^{(w)}(x_I) = w$ and $C_0^{(w)}(x) = \emptyset$ for all $x \in X_G, x \neq x_I$, $C_{2i}^{(w)} \Longrightarrow C_{2i+1}^{(w)}$ and $C_{2i+1}^{(w)} \vdash C_{2i+2}^{(w)}$, for all $i \geqslant 0$. By the previous definitions, each configuration $C_i^{(w)}$ is uniquely determined by the configuration $C_{i-1}^{(w)}$. In other terms, each computation in an AHNEP is deterministic. A computation as above immediately halts if one of the following two conditions holds:

(i) There exists a configuration in which the set of words existing in the output node x_O is nonempty. In this case, the computation is said to be an *accepting computation*.
(ii) There exist two consecutive identical configurations.

In the aforementioned cases the computation is said to be finite. The language accepted by Γ is

$$L_{\text{acc}}(\Gamma) = \{w \in V^* \mid \text{ the computation of } \Gamma \text{ on w is an accepting one}\}.$$

3. Size complexity of complete GHNEPs

Let L be a language generated by a complete GHNEP. We define

$$size(L) = \min\{size(\Gamma) \mid L = L_{\text{gen}}(\Gamma)\}.$$

The first natural problem concerns the existence of a constant upper bound for the size of any language generated by a GHNEP. The next theorem shows that this is not the case.

Theorem 1. *The measure size is connected, that is for any $n \geqslant 1$ there exists a language L_n such that $size(L_n) = n$.*

Proof. We consider the regular language $L_n = a_1^+ a_2^+ \dots a_n^+$ for any $n \geqslant 1$. Let Γ be a complete homogeneous GHNEP with n nodes $x_1, x_2, \dots, x_n$ and

$$
\begin{array}{ll}
M_{x_i} = \{\varepsilon \to a_i, \varepsilon \to a_{i+1}\}, \quad 1 \leqslant i \leqslant n-1, & M_{x_n} = \{\varepsilon \to a_n\}, \\
PI_{x_i} = \{a_1, a_2, \dots, a_i\}, \quad 1 \leqslant i \leqslant n, & FI_{x_i} = \{a_{i+1}, \dots, a_n\}, \quad 1 \leqslant i \leqslant n, \\
PO_{x_i} = PI_{x_i} \cup \{a_{i+1}\}, \quad 1 \leqslant i \leqslant n-1, & PO_{x_n} = \emptyset, \\
FO_{x_i} = \emptyset, \quad 1 \leqslant i \leqslant n-1, & FO_{x_n} = \{a_1, a_2, \dots, a_n\}, \\
C_0(x_1) = \{a_1\}, & C_0(x_i) = \emptyset, \ 2 \leqslant i \leqslant n, \\
\alpha(x_i) = r, \quad 1 \leqslant i \leqslant n. & \beta(x_i) = (1), \ 1 \leqslant i \leqslant n.
\end{array}
$$

It is obvious that $L_{\text{gen}}(\Gamma) = L_n$, hence $size(L_n) \leqslant n$.

Conversely, let us suppose that $L_n = L_{\text{gen}}(\Gamma')$ for some GHNEP Γ' of size at most $n-1$. By the pigeonhole principle, there exists a node in Γ' in which there are many arbitrarily inserted symbols which will eventually be transformed in two distinct symbols, say a_i and a_j with $1 \leqslant i < j \leqslant n$. It follows that Γ' will generate also words having a_j before a_i which is contradictory. □

Note that the languages L_n used in the above proof are defined over alphabets depending on n. Does the statement hold anymore for alphabets of a fixed size? If yes, which is this size?

The next result obtained in [17] is rather surprising since the size of the GHNEP, hence its underlying structure, does not depend on the number of states of the given automaton. In other words, this structure is common to all regular languages over the same alphabet, no matter the state complexity of the automata recognizing them. Furthermore, all words of the same length are generated simultaneously.

Theorem 2 (*Martin-Vide et al. [17]*). *Any regular language L over an alphabet with n symbols can be generated by a complete GHNEP of size $2n+3$. Hence, $size(L) \leqslant 2card(alph(L)) + 3$.*

Obviously, the GHNEP constructed according to the proof of the above theorem in [17] which generates a given regular language L depends on the number of states of the finite automaton defining L as well, but the underlying graph remains the same for all regular languages over the alphabet of L. For instance, if the number of states of an automaton accepting L is m, then the total number of symbols of the constructed GHNEP alphabet is $2n + 2nm + m$ while the total number of evolutionary rules is at most $2nm + 2n + 1$. However, for $n \geqslant 3$, one can reduce this size to $n + 5$.

Theorem 3. *For any regular language L, $size(L) \leqslant \min(2card(alph(L)) + 3, card(alph(L)) + 5)$.*

Proof. Let $A = (Q, V, \delta, q_0, F)$ be a deterministic finite automaton (DFA) accepting L with $card(V) = n$. We construct the following complete GHNEP:

$$\Gamma = (U, K_{n+5}, \mathcal{N}, C_0, \alpha, \beta, x_O),$$

where the alphabet U is defined by $U = V \cup V' \cup \tilde{V} \cup Q \cup \{[as] \mid s \in Q, a \in V\}$, with $V' = \{a' \mid a \in V\}$ and $\tilde{V} = \{\tilde{a} \mid a \in V\}$. The set of nodes of the complete

Table 1

Node	M	PI	FI	PO	FO	C_0	α	β
x_1	$\{q \to [as] \mid \delta(q,a) = s\}$	Q	$U \setminus (Q \cup V)$	U	$\emptyset$	$\{q_0\}$	$*$	(4)
x_2	$\{\varepsilon \to a' \mid a \in V\}$	$\{[as] \mid a \in V, s \in Q\}$	$\emptyset$	U	$\emptyset$	$\emptyset$	r	(4)
$x_a, a \in V$	$\{[as] \to s \mid s \in Q\}$	$\{a'\}$	$U \setminus (V \cup \{a'\} \cup \{[aq] \mid q \in Q\}$	U	$\emptyset$	$\emptyset$	$*$	(4)
x_3	$\{a' \to a \mid a \in V\}$	$\{c' \mid c \in V\}$	$U \setminus (V \cup V' \cup Q)$	U	$\emptyset$	$\emptyset$	$*$	(4)
x_4	$\{q \to \varepsilon \mid q \in F\}$	F	$U \setminus (Q \cup V)$	U	$\emptyset$	$\{q_0\}$	$*$	(4)
x_O	$\emptyset$	U	$U \setminus V$	$\emptyset$	$\emptyset$	$\emptyset$	$*$	(4)

underlying graph is $\{x_1, x_2, x_3, x_4, x_O\} \cup \{x_a \mid a \in V\}$, and the other parameters are given in Table 1.

One can easily prove by induction that

(1) $\delta(q_0, x) = s$ with $s \in Q \setminus F$ if and only if $sx \in C_{8|x|}(x_1)$.

(2) $\delta(q_0, x) = s$ with $s \in F$ if and only if $sx \in C_{8|x|}(x_1) \cap C_{8|x|}(x_4)$.

Therefore, $L(A)$ is exactly the language generated by Γ. Note that the number of symbols is now $3n + (n+1)card(Q)$ while the number of rules is at most $3n + (2n+1)card(Q)$. □

We want to stress that the size of a regular language can be also linearly bounded by the number of states of the minimal DFA recognizing that language. More precisely,

Theorem 4 (*Martin-Vide and Mitrana [16]*). *For any regular language L accepted by an n-state minimal DFA, $size(L) \leqslant 2n + 2$.*

A natural problem arises: given a regular language L, is $size(L)$ algorithmically computable? We are not able to give a complete answer to this problem. However, we can state:

Theorem 5. *Given a regular language L one can algorithmically decide whether or not $size(L) = 1$.*

Proof. Clearly, given a regular language $L \subseteq U^*$, $size(L) = 1$ if and only if there exist a finite subset E of L and an alphabet $V \subseteq U$ such that exactly one of the next three conditions is satisfied:

$$(1)\ L = EV^*, \qquad (2)\ L = ш(E, V^*), \qquad (3)\ L = V^*E, \qquad (*)$$

where ш is the *shuffle* operation defined on two words $x, y \in V^*$ by $ш(x, y) = \{x_1y_1x_2y_2, \ldots, x_ny_n \mid n \geqslant 1, x_i, y_i \in V^*, x = x_1x_2, \ldots, x_n, y = y_1, y_2, \ldots, y_n\}$, and extended to languages by $ш(L_1, L_2) = \bigcup_{x \in L_1, y \in L_2} ш(x, y)$. Indeed, if $L = EV^*$ as above, then one can construct a GHNEP of size 1 generating L by taking E as the initial set of words and $\{\varepsilon \to a \mid a \in V\}$ as the set of rules applied to the right only of the single

processor. If $L = \perp\!\!\!\perp(E, V^*)$, then the above rules are applied at any position. Conversely, if $size(L) = 1$, there exists a GHNEP generating L which consists of one processor. The initial finite set of words in this node is E and L is of the form (1)–(3) in $(*)$ depending on the way of applying the rules of the node: to the right, at any position, to the left, respectively.

We start by showing that the first condition can be algorithmically checked. First, we recall several definitions from [22]. A nondeterministic finite automaton (NFA) with nondeterministic starting state (NNFA) is an NFA except that it has a set of starting states instead of just one starting state. Any computation starts from a state nondeterministically chosen from this set. For a DFA $A = (Q, V, \delta, s, F)$, $\gamma(A) = (Q, V, \delta^R, F, \{s\})$ is the NNFA with $\delta^R : Q \to 2^Q$ defined by $\delta^R(p, a) = \{q \mid \delta(q, a) = p\}$. For a NNFA $M = (Q, V, \eta, S, F)$, $\tau(M) = (Q', V, \eta', s', F')$ is the DFA in which Q' is the set of all parts of Q that are reachable from the starting state $s' = S$ after applying the standard subset construction technique on M; also η' and F' are specified by this technique. With these preparations we now recall the following result:

Theorem 6 (*Brozozowski [1], Yu [22]*). *If $A = (Q, V, \delta, s, F)$ is a DFA with all states in Q reachable from s, then $L(\tau(\gamma(A))) = L^R$ and $\tau(\gamma(A))$ is a minimum-state DFA.*

Clearly, a regular language L can be decomposed in the form $E \cdot V^*$ with E being a finite set, if it is accepted by a DFA satisfying the following properties:

(1) All its cycles are actually loops labelled by letters from V in some final states (possibly all).
(2) If a final state has a loop, it has one loop for each letter from V and no other edge going out.
(3) If a final state has no loop, then for each letter $a \in V$ there is an edge from this state to another final state labelled with a.

We call such a DFA an *ultimately looped* DFA. We now proceed by proving the following fact, which will then allow us to give a decision procedure for the question "Can a given language L be decomposed in the form $E \cdot V^*$ with E being a finite set?"

Fact 1. *A regular language L can be decomposed in the form $E \cdot V^*$ with E being a finite set, if and only if the minimum-state DFA accepting L is ultimately looped.*

Proof. We start out from a language L that can be decomposed as $E \cdot V^*$; E can be chosen in such a way that for any word w in E there is no word $x \in V^+$ such that $wx \in E$. For instance, if $E = \{w, wx\}$ for some words w, x, then E can be simply taken as $\{w\}$. For now we suppose that $alph(E)$ and V are disjoint; if this is not the case, we rename the common letters obtaining T and the new language $L' = E \cdot T^*$. After the construction below, we shall return to the original alphabet V provided that $alph(E)$ and V are not disjoint. We construct a DFA A, which accepts L^R. First an initial state is created and for every letter in V a loop around this state is added. Then for every word $w \in E$ we add a path for w^R consisting of new states only which ends in a final state. The automaton constructed so far might be nondeterministic. By the standard subset construction, we transform it into a deterministic one which is the desired automaton A. Clearly, all states of A are reachable from its initial state.

Now, according to Theorem 6, the automaton $A' = \tau(\gamma(A))$ is a minimum-state DFA accepting L. We investigate its transition graph. The operation γ just inverts all edges of the graph. Because of the special form of A, $\gamma(A)$ is an ultimately looped NFA. Since $\gamma(A)$ has exactly one final state and $alph(E)$ and V were assumed to be disjoint, A' is an ultimately looped DFA. Indeed, A' has a final state q with one loop for each letter from V and for any other final state s and a letter $a \in V$, there exists a transition from s to q labelled by a.

Now we return to the case when $alph(E)$ and V are not disjoint. We replace all labels of the edges in A' by the letters from which they originated. The new automaton, say A'' may not be deterministic anymore. We apply the standard subset construction and then the minimization algorithm. By the aforementioned form of A', the result will be an ultimately looped minimum-state automaton which concludes the proof of the fact. □

So the minimum-state DFA constructed for L is ultimately looped; since all minimum-state automata recognizing a given language are isomorphic we infer that the question "Can a given regular language L be decomposed in the form $E \cdot V^*$ with E being a finite set?" can be decided by checking whether or not the minimum-state DFA recognizing L is ultimately looped. Since the final condition can be algorithmically tested, we conclude that the above problem is decidable.

The third condition in $(*)$ can also be algorithmically checked since $L = V * E$ iff $L^R = EV*$. It remains to show how the second condition in $(*)$ can be algorithmically tested. First, we can algorithmically determine the possible alphabet V by $V = \{a \in U \mid pr_{\{a\}}(L)$ is infinite$\}$.

The set V can be computed since the projection of regular language is a regular language and the finiteness problem is decidable for regular languages. Let A be a DFA recognizing L; clearly all edges of A which belong to a cycle are labelled by letters from V. We define the finite set

$$F = \{w \in L \mid w \text{ is computed by a cycle-free path in } A\}.$$

Fact 2. *$L(A)$ can be written as $L(A) = ш(E, V^*)$ for some finite set E if and only if $L(A) = ш(F, V^*)$.*

Proof. It suffices to prove the "only if" part. Let us assume that $L(A) = ш(E, V^*)$ for some finite set E. As the shuffle operation is associative it follows that $ш(L(A), V^*) = L(A)$. Since $F \subseteq L(A)$ we infer that $ш(F, V^*) \subseteq L(A)$. For the converse inclusion, take a word $z \in L$; if z can be computed by a cycle-free path in A, then $z \in F$ and we are done. Assume now that $z \in L$ and all $w \in L$ that can be computed with less cycles than z are $ш(F, V^*)$. By cutting an elementary cycle in the computation of z we get that $z = x_1yx_2$ such that $y \in V^+$ and $x_1x_2 \in L$ can be computed with less cycles than z. By our supposition, $x_1x_2 \in ш(F, V^*)$, hence $z \in ш(F, V^*)$ as well, which concludes the proof of the fact and of the theorem. □

However, a complete answer to the question: "Is the size of a regular language computable?" remains an open problem. The same problem is completely solved for context-free languages by showing that the problem considered in Theorem 5 is not decidable for context-free languages.

Theorem 7. *One cannot algorithmically decide whether the size of a context-free language equals* 1.

Proof. The proof is a reduction to the Post-Correspondence Problem (PCP). We take two post-lists over $\{a, b\}$: $x = (x_1, x_2, \ldots, x_p)$, and $y = (y_1, y_2, \ldots, y_p)$, and construct the languages

$$L(\alpha) = \{ba^{i_1}ba^{i_2}\ldots ba^{i_k}c\alpha_{i_k}\alpha_{i_{k-1}}\cdots\alpha_{i_1} \mid k \geqslant 1, 1 \leqslant i_j \leqslant p,\ 1 \leqslant j \leqslant k\}, \alpha \in \{x, y\},$$

$$L_{\mathrm{PCP}}(x, y) = (L(x)\{c\}L^R(y) \cap \{w_1cw_2cw_2^Rcw_1^R \mid w_1, w_2 \in \{a, b\}^+\}).$$

It is known that $L = \{a, b, c\}^* \setminus L_{\mathrm{PCP}}(x, y)$ is context-free. We consider the context-free language $L\{\$\}$, where \$ is a new symbol, and assume that $size(L\{\$\}) = 1$; since $\$ \in L\{\$\}$ it follows that $L\{\$\} = \{a, b, c\}^*\{\$\}$, hence the given instance of the PCP has no solution. Conversely, if $PCP(x, y)$ has no solution, then $L\{\$\}$ is $\{a, b, c\}^*\{\$\}$, hence obviously $size(L\{\$\}) = 1$. In conclusion, one cannot decide whether or not $size(L\{\$\}) = 1$. □

As an immediate consequence, we state

Corollary 1. *The measure size is not computable for the family of context-free languages.*

Since each linear grammar can be transformed into an equivalent linear grammar with rules of the form $A \to aB$, $A \to Ba$, $A \to \varepsilon$ only, the proof of Theorem 3 can be adapted for linear grammars as well. Moreover, the statement remains valid for GHNEPs with other types of underlying structure.

Theorem 8. *Any regular and linear language L over an alphabet with n symbols can be generated by a complete/star/ring GHNEP whose size depends linearly on n, only.*

A natural problem arises: Is it possible to give a similar characterization of other families of languages in the Chomsky hierarchy? Surprisingly enough, the answer is affirmative even for the class of recursively enumerable languages.

Theorem 9 (*Csuhaj-Varju et al. [5]*). *Any recursively enumerable language L over an alphabet V can be generated by a complete or star GHNEP of size* $28 + 3card(V)$. *Hence,* $size(L) \leqslant 28 + 3card(alph(L))$.

This last result suggests the possibility of constructing a "universal" GHNEP with a fixed underlying structure for all recursively enumerable languages over a given alphabet. Furthermore, a similar result based on the proof of Theorem 9 from [5] can be proved for context-free languages too.

The minimal size of a complete or star GHNEP generating an arbitrary recursively enumerable language over a fixed alphabet remains to be further investigated. However,

Table 2

Node	M	PI	FI	PO	FO	C_0	α	β
N_1	$\{\varepsilon \to a, \varepsilon \to b, \varepsilon \to X\}$	$\emptyset$	$\emptyset$	$\{c\}$	$\emptyset$	$\{\varepsilon\}$	$*$	(1)
N_2	$\{X \to c\}$	$\{X\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$*$	(1)

we can state:

Theorem 10. (1) *The language generated by any GHNEP of size one is regular.*

(2) *There exist noncontext-free languages which can be generated by complete, homogeneous GHNEPs of size* 2.

(3) *There exist nonrecursive languages which can be generated by complete or star GHNEPs of size* 28.

(4) *The family of languages generated by complete or star GHNEPs having no deletion node coincides with the family of context-sensitive languages.*

Proof. All statements, except the second one, are proved in [5], where it is given a complete homogeneous GHNEP of size 2 generating a context-free nonlinear language. Ralf Stiebe communicated us a complete homogeneous GHNEP of size 2 generating a noncontext-free language [20] which is presented below in a simplified version. The alphabet of the network is $\{a, b, c, X\}$ and the two nodes are N_1, N_2. Either of them can be considered the output node. We consider N_1 as the output node. The other parameters are given in Table 2.

The nonempty words in the processor N_1 at some moment may contain one, two, or all three letters in the alphabet $\{a, b, c\}$. While the word is in N_1 arbitrarily many as or bs can be inserted. As soon as X is inserted, the word leaves N_1 and enters N_2 where X is replaced by c. Now, the words returns to N_1 and the process resumes. Let L be the language generated by this GHNEP of size 2. We now consider the language $L' = L \cap a^+b^+c^+ = \{a^n b^m c^p \mid n, m \geqslant p \geqslant 1\}$. We argue that this language is not context-free. To this aim we use Ogden's lemma [18]. We assume the contrary and take the word $a^n b^n c^n$ and distinguish (mark) all the positions occupied by c (all the occurrences of c). Since for any decomposition of a sufficiently long word $z = uvwxy \in L'$, the following two conditions must be satisfied:

(i) v or x contains at least one distinguished position (one occurrence of c),

(ii) $uv^i wx^i y \in L'$ for any $i \geqslant 1$,

we get a contradiction. Therefore, either L is not context-free. □

The smallest size of a noncontext-free language is 2. Which is the smallest size of a noncontext-sensitive language? The same question for nonrecursive language remains open.

The next result, proved in [5], seems to be interesting. It raises a new series of open problems: which is the smallest size of an elementary GHNEP generating a nonrecursive or noncontext-free language? Is still the size of such a GHNEP generating a regular or recursively enumerable language linearly bounded by the alphabet size?

Theorem 11 (*Csuhaj-Varju et al. [5]*). *Any recursively enumerable language can be generated by an elementary, complete or star GHNEP.*

We finish this section by pointing out an interesting result from [21] which states that any recursively enumerable language can be generated by a GHNEP with an undirected underlying graph with 58 nodes. In other words, if the complete (ring, star) structure of a GHNEP generating a language L is discarded, then the size of L can be bounded by a constant. Clearly, by Theorem 1, such a result cannot hold for complete or ring GHNEPs.

4. Solving NP-complete problems in linear time with AHNEPs with linearly bounded resources

AHNEPs may be used for solving problems in the following way. For any instance of the problem the computation in the associated AHNEP is finite. In particular, this means that there is no node processor specialized in insertions. In the initial configuration, the input node contains an encoding of the instance we want to solve. If the problem is a decision problem, then at the end of the computation, the output node provides all solutions of the problem encoded by words, if any, otherwise this node will never contain any word. If the problem requires a finite set of words, this set will be in the output node at the end of the computation. In other cases, the result is collected by specific methods which is indicated for each problem.

Despite their simplicity, these mechanisms are able to solve hard problems in polynomial time as stated in Theorem 16. However, as we shall see in the sequel, some well-known NP-complete problems can be solved in linear time. In [2] there is presented a linear solution for an NP-complete problem, namely the Bounded Post-Correspondence Problem (BPCP). The BPCP is a variant of a much celebrated computer science problem, the PCP known to be unsolvable in the unbounded case. An instance of the BPCP consists of an alphabet V, two lists of words over $Vu = (u_1, u_2, \ldots, u_n)$ and $v = (v_1, v_2, \ldots, v_n)$, and $K \leqslant n$. The problem asks whether or not a sequence $i_1, i_2, \ldots, i_k$, $k \leqslant K$, of positive integers exists, each between 1 and n, such that $u_{i_1}u_{i_2}, \ldots, u_{i_k} = v_{i_1}v_{i_2}, \ldots, v_{i_k}$.

Theorem 12 (*Castellanos et al. [2]*). *The bounded PCP can be solved by a complete AHNEP in size and time linearly bounded by the product of K and the length of the longest word of the two post lists.*

In [3], weaker variants are still able to solve in linear time another NP-complete problem, namely the "3-colorability problem". This problem is to decide whether each vertex in an undirected graph can be colored by using three colors (say red, blue, and green) in such a way that after coloring, no two vertices which are connected by an edge have the same color.

Theorem 13 (*Castellanos et al. [3]*). *The "3-colorability problem" can be solved in* $O(m+n)$ *time by a complete AHNEP of size* $7m + 2$, *where n is the number of vertices and m is the number of edges of the input graph.*

It is rather interesting that the underlying graph of the AHNEP proposed in [3] for solving this problem does not depend on the number of nodes of the given instance of the problem.

In other words, the same underlying structure may be used for solving any instance of the 3-colorability problem having the same number of edges but no matter the number of nodes. Again, as in the case of language generating, the others parameters of the network depend on both numbers, of nodes and edges, but they still remain linearly bounded by these numbers. For instance, the total number of symbols is $7n + m + 1$ while the total number of rules is $16m + 3n + 1$.

In [17], following the descriptive format for three NP-complete problems (maximum independence set problem, vertex cover problem, satisfiability problem) presented in [13], one presents a solution to the *Common Algorithmic Problem* (CAP): let S be a finite set and F be a family of subsets of S. Find the cardinality of a maximal subset of S which does not include any set belonging to F. Let n and m be the cardinality of S and F, respectively.

Theorem 14 (*Martin-Vide et al. [17]*). *Any instance of the CAP can be solved by a complete homogeneous AHNEP of size* $m + 2n + 2$ *in* $O(m + n)$ *time*.

The price paid for homogeneity is the following: the total number of symbols is $2m + 4n + 4$, but the total number of rules is rather high, that is $mn + 6n + 2 + \sum_{i=1}^{m} card(F_i)$. The same problem can be solved in a more economic way with non-homogeneous AHNEPs, namely:

Theorem 15. *Any instance of the CAP can be solved by a complete AHNEP of size* $m + n + 1$ *in* $O(m + n)$ *time*.

Now, the needed resources are: $m + 3n + 3$ symbols and $m + 3n + 1$ rules. The second solution provides also a faster AHNEP: it solves an instance in time $2m + 4n + 1$ while the homogeneous AHNEP from above solves the same instance in time $8m + 4n + 3$.

5. Final remarks

The reader is referred to [11,12] for the classical time and space complexity classes defined on the standard computing model of Turing machine. We define some computational complexity measures by using AHNEP as the computing model following [15]. To this aim we consider an AHNEP Γ and the language L accepted by Γ. The *time complexity* of the accepting computation $C_0^{(x)}, C_1^{(x)}, C_2^{(x)}, \ldots, C_m^{(x)}$ of Γ on $x \in L$ is denoted by $Time_\Gamma(x)$ and equals m. The time complexity of Γ is the partial function from **N** to **N**,

$$Time_\Gamma(n) = \max\{Time_\Gamma(x) \mid x \in L_{\mathrm{acc}}(\Gamma), |x| = n\}.$$

For a function $f : \mathbf{N} \longrightarrow \mathbf{N}$ we define

$$\mathbf{Time}_{\mathrm{AHNEP}}(f(n)) = \{L \mid L = L_{\mathrm{acc}}(\Gamma) \text{ for an AHNEP } \Gamma \text{ with } Time_\Gamma(n) \leqslant f(n) \text{ for some } n \geqslant n_0\}.$$

Moreover, we write $\mathbf{PTime}_{\mathrm{AHNEP}} = \bigcup_{k \geqslant 0} \mathbf{Time}_{\mathrm{AHNEP}}(n^k)$.

$$\mathbf{ExpTime}_{\mathrm{AHNEP}} = \bigcup_{k \geqslant 0} \mathbf{Time}_{\mathrm{AHNEP}}(2^{n^k}).$$

Now we recall a result from [15] which establishes a strong connection between the complexity classes defined on Turing machines and those defined on AHNEPs.

Theorem 16 (*Margenstern et al. [15]*). (1) $\mathbf{NP} = \mathbf{PTime}_{\mathrm{AHNEP}}$.

(2) $\mathbf{NEXPTIME} = \mathbf{ExpTime}_{\mathrm{AHNEP}}$.

It is worth mentioning that the underlying graph of the AHNEP constructed in the proof of this theorem is the complete graph K_p, where

$$p = 18 + 7(card(V) - 1) + card(Q) + (card(V) - 1)^2 + 2card(Q)(card(V) - 1).$$

Here Q and V are the state set and tape alphabet, respectively. As one can see, the number of nodes of Γ is bounded by a quadratic function depending on the number of states and symbols of the Turing machine M which is simulated. It is useful to note that also the total number of symbols used by Γ in the above simulation is bounded by a cubic function depending on the number of states and symbols of M. More precisely, this number is

$$4card(Q)(card(V) - 1)^2 + 2(card(V) - 1)^2 + 2card(Q)(card(V) - 1), + 10(card(V) - 1) + card(Q).$$

References

[1] J.A. Brzozowski, Canonical regular expressions and minimal state graphs for definite events, Mathematical Theory of Automata, MRI Symposia Series, vol. 12, Polytechnic Press, NY, 1962, pp. 529–561.

[2] J. Castellanos, C. Martín-Vide, V. Mitrana, J. Sempere, Solving NP-complete problems with networks of evolutionary processors, in: J. Mira, A. Prieto (Eds.), IWANN 2001, Lecture Notes in Computer Science, vol. 2084, Springer, Berlin, 2001, pp. 621–628.

[3] J. Castellanos, C. Martín-Vide, V. Mitrana, J. Sempere, Networks of evolutionary processors, Acta Inform. 39 (2003) 517–529.

[4] E. Csuhaj-Varjú, J. Dassow, J. Kelemen, Gh. Păun, Grammar Systems, Gordon and Breach, London, 1994.

[5] E. Csuhaj-Varjú, C. Martín-Vide, V. Mitrana, Hybrid networks of evolutionary processors: completeness results, Acta Informatica, in press.

[6] E. Csuhaj-Varjú, V. Mitrana, Evolutionary systems: a language generating device inspired by evolving communities of cells, Acta Inform. 36 (2000) 913–926.

[7] E. Csuhaj-Varjú, A. Salomaa, Networks of parallel language processors, in: Gh. Păun, A. Salomaa (Eds.), New Trends in Formal Languages, Lecture Notes in Computer Science, vol. 1218, Springer, Berlin, 1997, pp. 299–318.

[8] E. Csuhaj-Varjú, A. Salomaa, Networks of Watson-Crick D0L systems, in: M. Ito, T. Imaoka (Eds.), Proc. Internat. Conf. Words, Languages and Combinatorics III, World Scientific, Singapore, 2003, pp. 134–150.

[9] L. Errico, C. Jesshope, Towards a new architecture for symbolic processing, in: I. Plander (Ed.), Artificial Intelligence and Information-Control Systems of Robots '94, World Scientific, Singapore, 1994, pp. 31–40.

[10] S.E. Fahlman, G.E. Hinton, T.J. Seijnowski, Massively parallel architectures for AI: NETL, THISTLE and Boltzmann machines, in: Proc. AAAI National Conf. on AI, William Kaufman, Los Altos, 1983, pp. 109–113.

[11] J. Hartmanis, P.M. Lewis II, R.E. Stearns, Hierarchies of memory limited computations, in: Proc. Sixth Ann. IEEE Symp. on Switching Circuit Theory and Logical Design, 1965, pp. 179–190.

[12] J. Hartmanis, R.E. Stearns, On the computational complexity of algorithms, Trans. Amer. Math. Soc. 117 (1965) 533–546.
[13] T. Head, M. Yamamura, S. Gal, Aqueous computing: writing on molecules, in: Proc. of the Cong. on Evolutionary Computation, IEEE Service Center, Piscataway, NJ, 1999, pp. 1006–1010.
[14] W.D. Hillis, The Connection Machine, MIT Press, Cambridge, MA, 1985.
[15] M. Margenstern, V. Mitrana, M. Perez-Jimenez, Accepting hybrid networks of evolutionary processors, Proc. of DNA 10, LNCS, Springer, Berlin, in press.
[16] C. Martín-Vide, V. Mitrana, Networks of evolutionary processors: results and perspectives, submitted for publication, in: M. Gheorghe (Ed.), Molecular Computational Models: Unconventional Approaches, Idea Group Publishing, Hershey, in press.
[17] C. Martín-Vide, V. Mitrana, M. Perez-Jimenez, F. Sancho-Caparrini, Hybrid networks of evolutionary processors, in: Proc. of GECCO 2003, 2003, pp. 401–412.
[18] W. Ogden, A helpful result for proving inherent ambiguity, Math. Systems Theory 2 (1968) 191–194.
[19] D. Sankoff, et al., Gene order comparisons for phylogenetic inference: evolution of the mitochondrial genome, Proc. Natl. Acad. Sci. USA 89 (1992) 6575–6579.
[20] R. Stiebe, personal communication.
[21] R. Stiebe, Some Considerations for HNEPs, personal communication.
[22] S. Yu, Regular languages, in: G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages, Springer, Berlin, 1997.